Kernel Methods and Support Vector Machines

John Shawe-Taylor
Department of Computer Science
University College London
jst@cs.ucl.ac.uk

June, 2009

Aim:

The tutorial is intended to give a broad introduction to the kernel approach to pattern analysis. This will cover:

- Why linear pattern functions?
- Why kernel approach?
- How to plug and play with the different components of a kernel-based pattern analysis system?

What won't be included:

- Other approaches to Pattern Analysis
- Complete History
- Bayesian view of kernel methods
- More recent developments

OVERALL STRUCTURE

Part 1: Introduction to the Kernel methods approach.

Part 2: Projections and subspaces in the feature space.

Part 3: Other learning algorithms with the example of Support Vector Machines.

Part 4: Kernel design strategies.

PART 1 STRUCTURE

- Introduction to pattern analysis and brief history
- Kernel methods approach
- Worked example of kernel Ridge Regression
- Properties of kernels.

Pattern Analysis

- Data can exhibit regularities that may or may not be immediately apparent
  - exact patterns – eg motions of planets
  - complex patterns – eg genes in DNA
  - probabilistic patterns – eg market research
- Detecting patterns makes it possible to understand and/or exploit the regularities to make predictions
- Pattern analysis is the study of automatic detection of patterns in data

Defining patterns

- Exact patterns: non-trivial function f such that

$$f(\mathbf{x}) = 0$$

- Approximate patterns: f such that

$$f(\mathbf{x}) \approx 0$$

- Statistical patterns: f such that

$$\mathbb{E}_{\mathbf{x}}[f(\mathbf{x})] \approx 0$$

Pattern analysis algorithms

We would like algorithms to be:

- Computationally efficient – running time polynomial in the size of the data – often needs to be of a low degree
- Robust – able to handle noisy data, eg examples misclassified, noisy sensors or outputs only to a certain accuracy
- Statistical stability – able to distinguish between chance patterns and those characteristic of the underlying source of the data

Brief Historical Perspective

- Machine learning using neural like structures first considered seriously in 1960s with such systems as the Perceptron
  - Linear patterns
  - Simple learning algorithm
  - shown to be limited in complexity
- Resurrection of ideas in more powerful multi-layer perceptrons in 1980s
  - networks of perceptrons with continuous activation functions
  - very slow learning
  - limited statistical analysis

Kernel methods

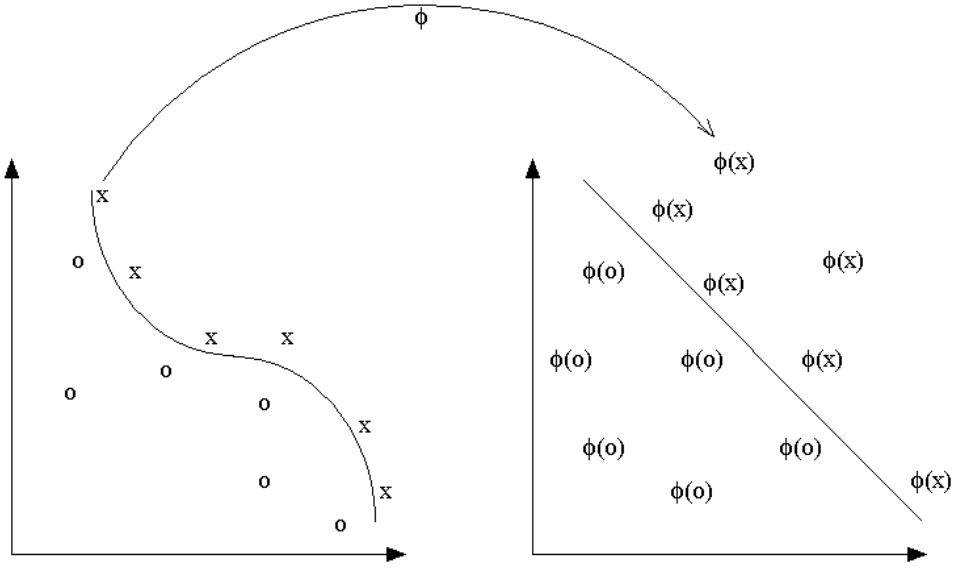
Kernel methods (re)introduced in 1990s with Support Vector Machines

- Linear functions but in high dimensional spaces equivalent to non-linear functions in the input space
- Statistical analysis showing large margin can overcome curse of dimensionality
- Extensions rapidly introduced for many other tasks other than classification

Kernel methods approach

- Data embedded into a Euclidean feature space
- Linear relations are sought among the images of the data
- Algorithms implemented so that only require inner products between vectors
- Embedding designed so that inner products of images of two points can be computed directly by an efficient 'short-cut' known as the kernel.

Kernel methods embedding



- The function ϕ embeds the data into a feature space where the non-linear pattern now appears linear. The kernel computes inner products in the feature space directly from the inputs.

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

Worked example: Ridge Regression

Consider the problem of finding a homogeneous real-valued linear function

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{x}'\mathbf{w} = \sum_{i=1}^{n} w_i x_i,$$

that best interpolates a given training set

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$$

of points $\mathbf{x}_i$ from $X \subseteq \mathbb{R}^n$ with corresponding labels y_i in $Y \subseteq \mathbb{R}$.

Possible pattern function

- Measures discrepancy between function output and correct output – squared to ensure always positive:

$$f_g((\mathbf{x}, y)) = (g(\mathbf{x}) - y)^2$$

  Note that the pattern function f_g is not itself a linear function, but a simple functional of the linear functions g.

- We introduce notation: matrix $\mathbf{X}$ has rows the m examples of S. Hence we can write

$$\xi = \mathbf{y} - \mathbf{X}\mathbf{w}$$

  for the vector of differences between $g(\mathbf{x}_i)$ and y_i.

Optimising the choice of g

Need to ensure flexibility of g is controlled – controlling the norm of $\mathbf{w}$ proves effective:

$$\min_{\mathbf{w}} \mathcal{L}_\lambda(\mathbf{w}, S) = \min_{\mathbf{w}} \lambda \|\mathbf{w}\|^2 + \|\xi\|^2,$$

where we can compute

$$\begin{aligned} \|\xi\|^2 &= \langle \mathbf{y} - \mathbf{X}\mathbf{w}, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle \\ &= \mathbf{y}'\mathbf{y} - 2\mathbf{w}'\mathbf{X}'\mathbf{y} + \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} \end{aligned}$$

Setting derivative of $\mathcal{L}_\lambda(\mathbf{w}, S)$ equal to 0 gives

$$\mathbf{X}'\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n)\,\mathbf{w} = \mathbf{X}'\mathbf{y}$$

Primal solution

The term primal is used for the explicit representation in the feature space:

- We get the primal solution weight vector:

$$\mathbf{w} = \left(\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}_n\right)^{-1} \mathbf{X}'\mathbf{y}$$

- and regression function

$$g(\mathbf{x}) = \mathbf{x}'\mathbf{w} = \mathbf{x}' \left(\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}_n\right)^{-1} \mathbf{X}'\mathbf{y}$$

Dual solution

A dual solution expresses the weight vector as a linear combination of the training examples:

$$\mathbf{X}'\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \mathbf{X}'\mathbf{y} \quad \text{implies}$$

$$\mathbf{w} = \frac{1}{\lambda}\left(\mathbf{X}'\mathbf{y} - \mathbf{X}'\mathbf{X}\mathbf{w}\right) = \mathbf{X}'\frac{1}{\lambda}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) = \mathbf{X}'\alpha,$$

where

$$\alpha = \frac{1}{\lambda}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) \tag{1}$$

or equivalently

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i$$

The vector α is the dual solution.

Dual solution

Substituting $\mathbf{w} = \mathbf{X}'\alpha$ into equation (1) we obtain:

$$\lambda\alpha = \mathbf{y} - \mathbf{X}\mathbf{X}'\alpha$$

implying

$$(\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}_m)\,\alpha = \mathbf{y}$$

This means the dual solution can be computed as:

$$\alpha = (\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}_m)^{-1}\,\mathbf{y}$$

with the regression function

$$g(\mathbf{x}) = \mathbf{x}'\mathbf{w} = \mathbf{x}'\mathbf{X}'\alpha = \left\langle \mathbf{x}, \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \right\rangle = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Key ingredients of dual solution

Step 1: Compute

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}'$ that is $\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

Step 2: Evaluate on new point $\mathbf{x}$ by

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Important observation: Both steps only involve inner products between input data points

Applying the 'kernel trick'

Since the computation only involves inner products, we can substitute for all occurrences of $\langle \cdot, \cdot \rangle$ a kernel function κ that computes:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and we obtain an algorithm for ridge regression in the feature space F defined by the mapping

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in F$$

Note if ϕ is the identity this has no effect.

A simple kernel example

The simplest non-trivial kernel function is the quadratic kernel:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^2$$

involving just one extra operation. But surprisingly this kernel function now corresponds to a complex feature mapping:

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}'\mathbf{z})^2 = \mathbf{z}'(\mathbf{x}\mathbf{x}')\mathbf{z} \\ &= \langle \mathrm{vec}(\mathbf{z}\mathbf{z}'), \mathrm{vec}(\mathbf{x}\mathbf{x}') \rangle \end{aligned}$$

where $\mathrm{vec}(A)$ stacks the columns of the matrix A on top of each other. Hence, κ corresponds to the feature mapping

$$\phi : \mathbf{x} \longmapsto \mathrm{vec}(\mathbf{x}\mathbf{x}')$$

Implications of the kernel trick

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$ pixels.

- By using the quadratic kernel we implement the regression function in a $1,000,000$ dimensional space

- but actually using less computation for the learning phase than we did in the original space – inverting a 1000×1000 matrix instead of a 1024×1024 matrix.

Implications of kernel algorithms

- Can perform linear regression in very high-dimensional (even infinite dimensional) spaces efficiently.

- This is equivalent to performing non-linear regression in the original input space: for example quadratic kernel leads to solution of the form

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle^2$$

  that is a quadratic polynomial function of the components of the input vector $\mathbf{x}$.

- Using these high-dimensional spaces must surely come with a health warning, what about the curse of dimensionality?

Defining kernels

- Natural to consider defining kernels for your data. Clearly, kernel must be symmetric and satisfy

$$\kappa(\mathbf{x}, \mathbf{x}) > 0$$

- BUT not every function satisfying these conditions is a kernel.

- Commonly used kernel is Gaussian kernel:

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|}{2\sigma^2}\right)$$

  corresponds to infinite dimensional feature space.

Means and distances

Suppose we are given a kernel function:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and a training set S, what can we estimate?

- Consider some vector

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i)$$

we have

$$\|\mathbf{w}\|^2 = \left\langle \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i), \sum_{j=1}^{m} \alpha_j \phi(\mathbf{x}_j) \right\rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

Means and distances

- Hence, we can normalise data in the feature space:

$$\phi(\mathbf{x}) \mapsto \hat{\phi}(\mathbf{x}) = \frac{\phi(\mathbf{x})}{\|\phi(\mathbf{x})\|}$$

since we can compute the corresponding kernel $\hat{\kappa}$ by

$$\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \left\langle \frac{\phi(\mathbf{x})}{\|\phi(\mathbf{x})\|}, \frac{\phi(\mathbf{z})}{\|\phi(\mathbf{z})\|} \right\rangle = \frac{\kappa(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa(\mathbf{x}, \mathbf{x})\kappa(\mathbf{z}, \mathbf{z})}}$$

Means and distances

- Given two vectors:

$$\mathbf{w}_a = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i) \text{ and } \mathbf{w}_b = \sum_{i=1}^{m} \beta_i \phi(\mathbf{x}_i)$$

we have

$$\mathbf{w}_a - \mathbf{w}_b = \sum_{i=1}^{m} (\alpha_i - \beta_i) \phi(\mathbf{x}_i)$$

so we can compute the distance between $\mathbf{w}_a$ and $\mathbf{w}_b$ as

$$d(\mathbf{w}_a, \mathbf{w}_b) = \|\mathbf{w}_a - \mathbf{w}_b\|$$

Means and distances

- For example the norm of the mean of a sample is given by

$$\|\phi_S\| = \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(\mathbf{x}_i) \right\| = \frac{1}{m} \sqrt{\mathbf{j}'\mathbf{K}\mathbf{j}}$$

  where $\mathbf{j}$ is the all ones vector.

- Hence, expected squared distance to the mean of a sample is:

$$\begin{aligned} \hat{\mathbb{E}}[\|\phi(\mathbf{x}) - \phi_S\|^2] &= \frac{1}{m} \sum_{i=1}^{m} \kappa(\mathbf{x}_i, \mathbf{x}_i) - \langle \phi_S, \phi_S \rangle \\ &= \frac{1}{m} \mathrm{tr}(\mathbf{K}) - \frac{1}{m^2} \mathbf{j}'\mathbf{K}\mathbf{j} \end{aligned}$$

Means and distances

- Consider centering the sample, i.e. moving the origin to the sample mean: this will result in

$$\|\phi_S\|^2 = \frac{1}{m^2}\mathbf{j}'\mathbf{K}\mathbf{j} = 0$$

  in the new coordinate system, while the lhs of previous equation is unchanged by centering. Hence, centering minimises the trace.

- Centering is achieved by transformation:

$$\phi(\mathbf{x}) \mapsto \hat{\phi}(\mathbf{x}) = \phi(\mathbf{x}) - \phi_S$$

Means and distances

- What is effect on kernel and kernel matrix?

$$\begin{aligned}\hat{\kappa}(\mathbf{x},\mathbf{z}) &= \langle\hat{\phi}(\mathbf{x}),\hat{\phi}(\mathbf{z})\rangle \\ &= \kappa(\mathbf{x},\mathbf{z}) - \frac{1}{m}\sum_{i=1}^{m}(\kappa(\mathbf{x},\mathbf{x}_i)+\kappa(\mathbf{z},\mathbf{x}_i)) + \frac{1}{m^2}\mathbf{j}'\mathbf{K}\mathbf{j}\end{aligned}$$

- Hence we can implement the centering of a kernel matrix by

$$\hat{\mathbf{K}} = \mathbf{K} - \frac{1}{m}(\mathbf{j}\mathbf{j}'\mathbf{K} + \mathbf{K}\mathbf{j}\mathbf{j}') + \frac{1}{m^2}(\mathbf{j}'\mathbf{K}\mathbf{j})\mathbf{j}\mathbf{j}'$$

Simple novelty detection

- Consider putting a ball round the centre of mass ϕ_S of radius sufficient to contain all the data:

$$\|\phi(\mathbf{x}) - \phi_S\| > \max_{1 \leq i \leq m} \|\phi(\mathbf{x}_i) - \phi_S\|$$

- Give a kernel expression for this quantity.

OVERALL STRUCTURE

Part 1: Introduction to the Kernel methods approach.

Part 2: Projections and subspaces in the feature space.

Part 3: Other learning algorithms with the example of Support Vector Machines.

Part 4: Kernel design strategies.

Part 2 structure

- Simple classification algorithm
- Fisher discriminant analysis.
- Principal components analysis.

Simple classification algorithm

- Consider finding the centres of mass of positive and negative examples and classifying a test point by measuring which is closest

$$h(\mathbf{x}) = \operatorname{sgn}\left(\|\phi(\mathbf{x}) - \phi_{S_-}\|^2 - \|\phi(\mathbf{x}) - \phi_{S_+}\|^2\right)$$

- we can express as a function of kernel evaluations

$$h(\mathbf{x}) = \operatorname{sgn}\left(\frac{1}{m_+}\sum_{i=1}^{m_+}\kappa(\mathbf{x},\mathbf{x}_i) - \frac{1}{m_-}\sum_{i=m_++1}^{m}\kappa(\mathbf{x},\mathbf{x}_i) - b\right),$$

where

$$b = \frac{1}{2m_+^2}\sum_{i,j=1}^{m_+}\kappa(\mathbf{x}_i,\mathbf{x}_j) - \frac{1}{2m_-^2}\sum_{i,j=m_++1}^{m}\kappa(\mathbf{x}_i,\mathbf{x}_j)$$

Simple classification algorithm

- equivalent to dividing the space with a hyperplane perpendicular to the line half way between the two centres with vector given by

$$\mathbf{w} = \frac{1}{m^+}\sum_{i=1}^{m^+}\phi(\mathbf{x}_i) - \frac{1}{m^-}\sum_{i=m^+ +1}^{m}\phi(\mathbf{x}_i)$$

- Function is the difference in likelihood of the Parzen window density estimators for positive and negative examples

- We will see some examples of the performance of this algorithm in Part 3.

Variance of projections

- Consider projections of the datapoints $\phi(\mathbf{x}_i)$ onto a unit vector direction $\mathbf{v}$ in the feature space: average is given by

$$\mu_{\mathbf{v}} = \hat{\mathrm{E}}\left[\|P_{\mathbf{v}}(\phi(\mathbf{x}))\|\right] = \hat{\mathrm{E}}\left[\mathbf{v}'\phi(\mathbf{x})\right] = \mathbf{v}'\phi_S$$

  of course this is 0 if the data has been centred.

- average squared is given by

$$\hat{\mathrm{E}}\left[\|P_{\mathbf{v}}(\phi(\mathbf{x}))\|^2\right] = \hat{\mathrm{E}}\left[\mathbf{v}'\phi(\mathbf{x})\phi(\mathbf{x})'\mathbf{v}\right] = \frac{1}{m}\mathbf{v}'\mathbf{X}'\mathbf{X}\mathbf{v}$$

Variance of projections

- Now suppose $\mathbf{v}$ has the dual representation $\mathbf{v} = \mathbf{X}'\alpha$. Average is given by

$$\mu_{\mathbf{v}} = \frac{1}{m}\alpha'\mathbf{X}\mathbf{X}'\mathbf{j} = \frac{1}{m}\alpha'\mathbf{K}\mathbf{j}$$

- average squared is given by

$$\frac{1}{m}\mathbf{v}'\mathbf{X}'\mathbf{X}\mathbf{v} = \frac{1}{m}\alpha'\mathbf{X}\mathbf{X}'\mathbf{X}\mathbf{X}'\alpha = \frac{1}{m}\alpha'\mathbf{K}^2\alpha$$

- Hence, variance in direction $\mathbf{v}$ is given by

$$\sigma^2_{\mathbf{v}} = \frac{1}{m}\alpha^2\mathbf{K}^2\alpha - \frac{1}{m^2}(\alpha'\mathbf{K}\mathbf{j})^2$$

Fisher discriminant

- The Fisher discriminant is a thresholded linear classifier:

$$f(\mathbf{x}) = \mathrm{sgn}(\langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b$$

where $\mathbf{w}$ is chosen to maximise the quotient:

$$J(\mathbf{w}) = \frac{(\mu_{\mathbf{w}}^{+} - \mu_{\mathbf{w}}^{-})^2}{(\sigma_{\mathbf{w}}^{+})^2 + (\sigma_{\mathbf{w}}^{-})^2}$$

- As with Ridge regression is makes sense to regularise if we are working in high-dimensional kernel spaces, so maximise

$$J(\mathbf{w}) = \frac{(\mu_{\mathbf{w}}^{+} - \mu_{\mathbf{w}}^{-})^2}{(\sigma_{\mathbf{w}}^{+})^2 + (\sigma_{\mathbf{w}}^{-})^2 + \lambda \|\mathbf{w}\|^2}$$

Fisher discriminant

- Using the results we now have we can substitue dual expressions for all of these quantities and solve using lagrange multipliers.

- The resulting classifier has dual variables

$$\alpha = (\mathbf{BK} + \lambda \mathbf{I})^{-1}\mathbf{y}$$

where $\mathbf{B} = \mathbf{D} - \mathbf{C}$ with

$$\mathbf{C}_{ij} = \begin{cases} 2m^-/(mm^+) & \text{if } y_i = 1 = y_j \\ 2m^+/(mm^-) & \text{if } y_i = -1 = y_j \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mathbf{D} = \begin{cases} 2m^-/m & \text{if } i = j \text{ and } y_i = 1 \\ 2m^+/m & \text{if } i = j \text{ and } y_i = -1 \\ 0 & \text{otherwise} \end{cases}$$

and $b = 0.5\alpha\mathbf{K}\mathbf{t}$ with

$$\mathbf{t}_i = \begin{cases} 1/m^+ & \text{if } y_i = 1 \\ 1/m^- & \text{if } y_i = -1 \\ 0 & \text{otherwise} \end{cases}$$

giving a decision function

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) - b\right)$$

Overview of remainder of tutorial

- Plug and play aspects of kernel methods:

  Data $\rightarrow$ kernel $\rightarrow$ preprocessing $\rightarrow$ pattern analysis

- Part 2: preprocessing: for example normalisation, projection into subspaces, kernel PCA, kernel CCA, etc.

- Part 3: pattern analysis: support vector machines, novely detection, support vector regression.

- Part 4: kernel design: properties of kernels, kernels for text, string kernels.

Preprocessing

- Corresponds to feature selection, or learning the feature space

- Note that in kernel methods the feature space is only determined up to orthogonal transformations (change of basis):

$$\hat{\phi}(\mathbf{x}) = \mathbf{U}\phi(\mathbf{x})$$

  for some orthogonal transformation $\mathbf{U}$ ($\mathbf{U}'\mathbf{U} = \mathbf{I} = \mathbf{U}\mathbf{U}'$), then

$$\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \langle \mathbf{U}\phi(\mathbf{x}), \mathbf{U}\phi(\mathbf{z}) \rangle = \phi(\mathbf{x})'\mathbf{U}'\mathbf{U}\phi(\mathbf{z}) = \phi(\mathbf{x})'\phi(\mathbf{z}) = \kappa(\mathbf{x}, \mathbf{z})$$

- so feature selection is eqivalent to subspace projection in kernel defined feature spaces

Subspace methods

- Principal components analysis: choose directions to maximise variance in the training data
- Canonical correlation analysis: choose directions to maximise correlations between two different views of the same objects
- Gram-Schmidt: greedily choose directions according to largest residual norms
- Partial least squares: greedily choose directions with maximal covariance with the target (will not cover this)

In all cases we need kernel versions in order to apply these methods in high-dimensional kernel defined feature spaces

Principal Components Analysis

- PCA is a subspace method – that is it involves projecting the data into a lower dimensional space.

- Subspace is chosen to ensure maximal variance of the projections:

$$\mathbf{w} = \text{argmax}_{\mathbf{w}:\|\mathbf{w}\|=1} \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w}$$

- This is equivalent to maximising the Raleigh quotient:

$$\frac{\mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w}}{\mathbf{w}'\mathbf{w}}$$

Principal Components Analysis

- We can optimise using Lagrange multipliers in order to remove the contraints:

$$L(\mathbf{w}, \lambda) = \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} - \lambda\mathbf{w}'\mathbf{w}$$

  taking derivatives wrt $\mathbf{w}$ and setting equal to 0 gives:

$$\mathbf{X}'\mathbf{X}\mathbf{w} = \lambda\mathbf{w}$$

  implying $\mathbf{w}$ is an eigenvalue of $\mathbf{X}'\mathbf{X}$.

- Note that

$$\lambda = \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} = \sum_{i=1}^{m} \langle \mathbf{w}, \mathbf{x}_i \rangle^2$$

Principal Components Analysis

- So principal components analysis performs an eigenvalue decomposition of $\mathbf{X}'\mathbf{X}$ and projects into the space spanned by the first k eigenvectors

- Captures a total of

$$\sum_{i=1}^{k} \lambda_i$$

of the overall variance:

$$\sum_{i=1}^{m} \|\mathbf{x}_i\|^2 = \sum_{i=1}^{n} \lambda_i = \mathrm{tr}(\mathbf{K})$$

Kernel PCA

- We would like to find a dual representation of the principal eigenvectors and hence of the projection function.

- Suppose that $\mathbf{w}, \lambda \neq 0$ is an eigenvector/eigenvalue pair for $\mathbf{X}'\mathbf{X}$, then $\mathbf{X}\mathbf{w}, \lambda$ is for $\mathbf{X}\mathbf{X}'$:

$$(\mathbf{X}\mathbf{X}')\mathbf{X}\mathbf{w} = \mathbf{X}(\mathbf{X}'\mathbf{X})\mathbf{w} = \lambda\mathbf{X}\mathbf{w}$$

- and vice versa $\alpha, \lambda \rightarrow \mathbf{X}'\alpha, \lambda$

$$(\mathbf{X}'\mathbf{X})\mathbf{X}'\alpha = \mathbf{X}'(\mathbf{X}\mathbf{X}')\alpha = \lambda\mathbf{X}'\alpha$$

- Note that we get back to where we started if we do it twice.

Kernel PCA

- Hence, 1-1 correspondence between eigenvectors corresponding to non-zero eigenvalues, but note that if $\|\alpha\| = 1$

$$\|\mathbf{X}'\alpha\|^2 = \alpha'\mathbf{X}\mathbf{X}'\alpha = \alpha'\mathbf{K}\alpha = \lambda$$

so if $\alpha^i, \lambda_i, i = 1, \ldots, k$ are first k eigenvectors/values of $\mathbf{K}$

$$\frac{1}{\sqrt{\lambda_i}}\alpha^i$$

are dual representations of first k eigenvectors $\mathbf{w}^1, \ldots, \mathbf{w}^k$ of $\mathbf{X}'\mathbf{X}$ with same eigenvalues.

- Computing projections:

$$\langle \mathbf{w}^i, \phi(\mathbf{x}) \rangle = \frac{1}{\sqrt{\lambda_i}} \langle \mathbf{X}'\alpha^i, \phi(\mathbf{x}) \rangle = \frac{1}{\sqrt{\lambda_i}} \sum_{j=1}^{m} \alpha_j^i \kappa(\mathbf{x}_i, \mathbf{x})$$

OVERALL STRUCTURE

Part 1: Introduction to the Kernel methods approach.

Part 2: Projections and subspaces in the feature space.

Part 3: Other learning algorithms with the example of Support Vector Machines.

Part 4: Kernel design strategies.

Part 3 structure

- Perceptron algorithm
- Generalisation of SVMs
- Support Vector Machine Optimisation
- Novelty detection

Kernel algorithms

- Have already seen three kernel based algorithms:
  - Ridge regression
  - Fisher discriminant
  - Simple novelty detector

- Key properties that enable an algorithm to be *kernelised*:
  - Must reduce to estimating a linear function in the feature space
  - Weight vector must be in the span of the training examples
  - Algorithm to find dual coefficients only involves inner products of training data

- Very simple example: perceptron algorithm

Perceptron algorithm

- initialise $\mathbf{w} \leftarrow 0$
- repeat if for some example: $y_i \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle \leq 0$

$$\mathbf{w} \longleftarrow \mathbf{w} + y_i \phi(\mathbf{x}_i)$$

- Clearly dual version:
  - initialise $\alpha_i \leftarrow 0$ for all i
  - update: $\alpha_i \leftarrow \alpha_i + y_i$
  - Note can evaluate as for ridge regression:

$$\langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$$

- Dual version by Aizerman et al. (1964) but tends to overfit

Margin Perceptron algorithm

- Margin version if replace test by

$$y_i \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle \leq \tau$$

- Set $\tau = 1$ – in 1964 one parameter away from a kernel classification algorithm able to generalise in high dimensions

Support Vector Machines (SVM)

- SVM seeks linear function in a feature space defined implicitly via a kernel κ:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

  that optimises a bound on the generalisation.

- Several bounds on the performance of SVMs exist all use the margin to give an empirically defined complexity: data-dependent structural risk minimisation

- Tightest is the PAC-Bayes bound

Margins in SVMs

- Critical to the bound will be the margin of the classifier

$$\gamma(\mathbf{x}, y) = yg(\mathbf{x}) = y(\langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b) :$$

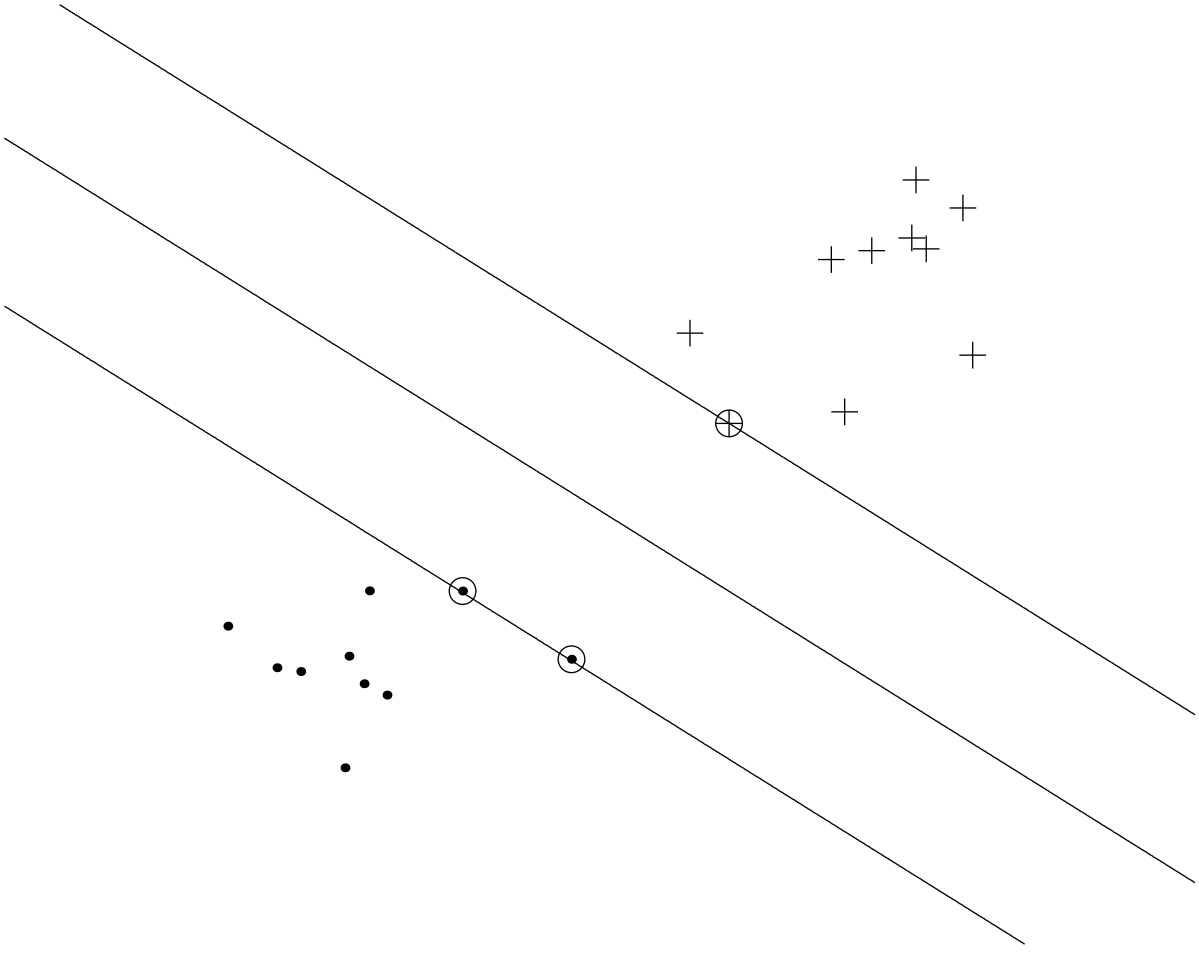
  positive if correctly classified, and measures distance from the separating hyperplane when the weight vector is normalised.

- The margin of a linear function g is

$$\gamma(g) = \min_i \gamma(\mathbf{x}_i, y_i)$$

  though this is frequently increased to allow some 'margin errors'.

Margins in SVMs



Form of the SVM bound

- If we define the inverse of the KL by

$$\mathrm{KL}^{-1}(q, A) = \max\{p : \mathrm{KL}(q\|p) \leq A\}$$

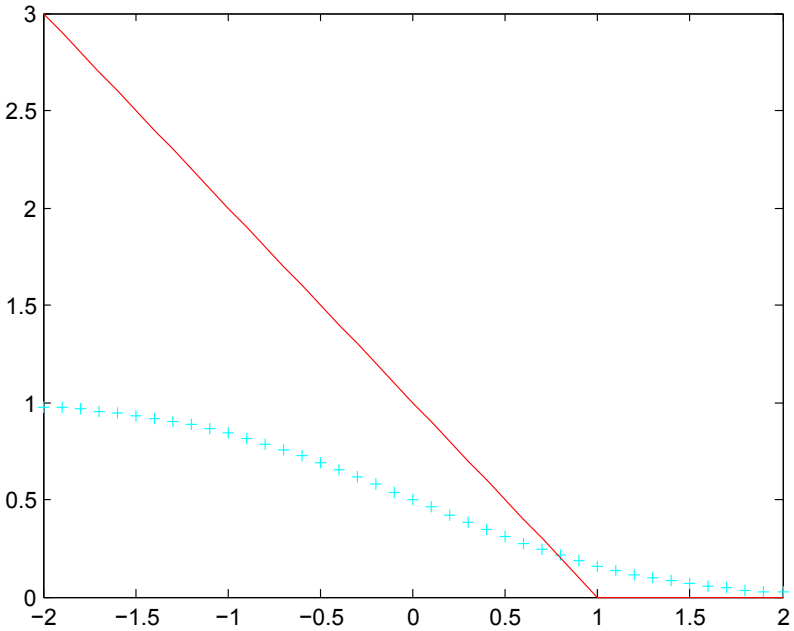
then have with probability at least $1 - \delta$ for all μ

$$Pr\left(\langle \mathbf{w}, \phi(\mathbf{x})\rangle \neq y\right) \quad \leq$$

$$2 \min_{\mu} \mathrm{KL}^{-1}\left(\mathbb{E}_m[\tilde{F}(\mu\gamma(\boldsymbol{x}, y))], \frac{\mu^2/2 + \ln \frac{m+1}{\delta}}{m}\right)$$

where $\tilde{F}(t) = 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-x^2/2} \mathrm{d}x$

Slack variable conversion



Bound and slack variable used in optimisation

Gives SVM Optimisation

- Primal form:

$$\begin{aligned} & \min_{\boldsymbol{w},\xi_i} \left[\tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^m \xi_i\right] \\ \text{s.t.} \quad & y_i \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i) \geq 1 - \xi_i \qquad i = 1, \ldots, m \\ & \xi_i \geq 0 \qquad i = 1, \ldots, m \end{aligned}$$

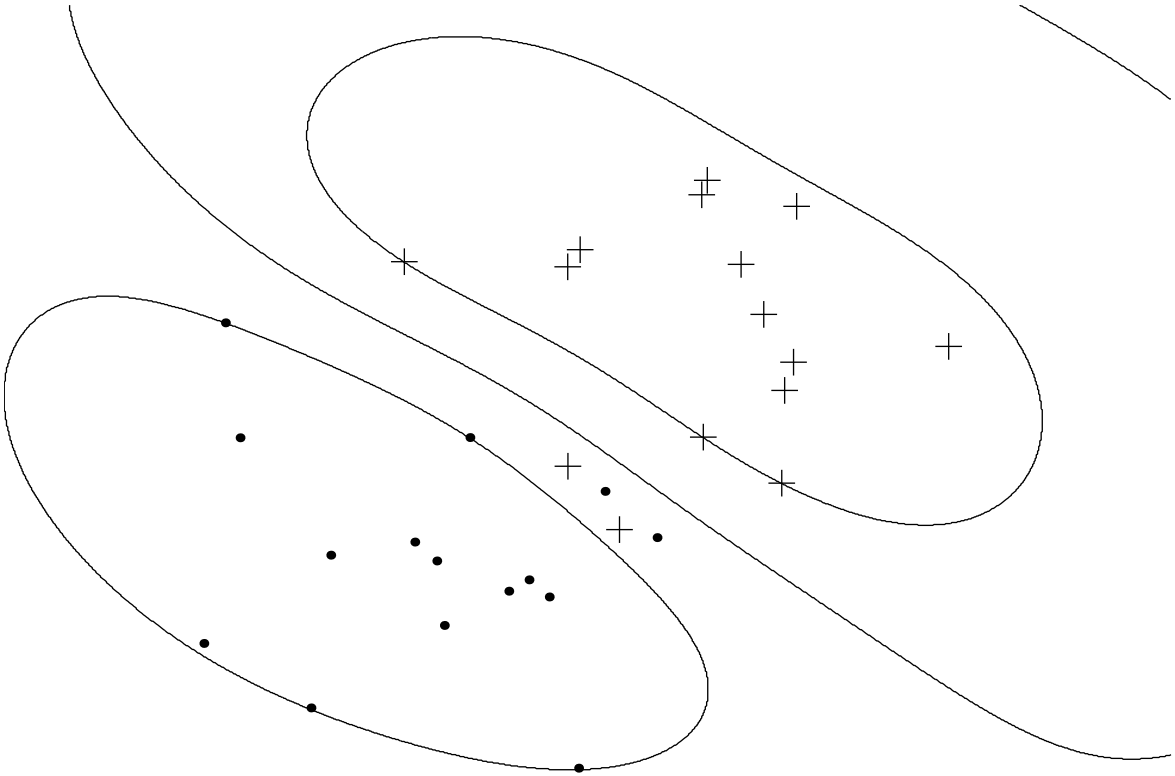
- Dual form:

$$\begin{aligned} & \max_{\boldsymbol{\alpha}} \left[\sum_{i=1}^m \alpha_i - \tfrac{1}{2}\sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)\right] \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad i = 1, \ldots, m \end{aligned}$$

where $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle$ and $\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle = \sum_{i=1}^m \alpha_i y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x})$.

Dual form of the SVM problem

Decision boundary and γ margin for 1-norm svm with a gaussian kernel:



Novelty detection

We can also motivate novelty detection by a similar analysis as that for SVM: consider a hypersphere centred at $\mathbf{c}$ of radius r and the function g:

$$g(\mathbf{x}) = \begin{cases} 0, & \text{if } \|\mathbf{c} - \phi(\mathbf{x})\| \leq r; \\ (\|\mathbf{c} - \phi(\mathbf{x})\|^2 - r^2)/\gamma, & \text{if } r^2 \leq \|\mathbf{c} - \phi(\mathbf{x})\|^2 \leq r^2 + \gamma; \\ 1, & \text{otherwise.} \end{cases}$$

with probability at least $1 - \delta$

$$\mathbb{E}[g(\mathbf{x})] \leq \hat{\mathbb{E}}[g(\mathbf{x})] + \frac{6R^2}{\gamma\sqrt{m}} + 3\sqrt{\frac{\ln(2/\delta)}{2m}}$$

Note that tension is between creating a tight bound and defining a small sphere.

Novelty detection

Let

$$\xi_i = \left(\|\mathbf{c} - \phi(\mathbf{x})\|^2 - r^2\right)_+$$

so that

$$\hat{\mathbb{E}}[g(\mathbf{x})] \leq \frac{1}{\gamma m}\|\xi\|_1$$

Treating γ as fixed we minimise the bound by minimising $\|\xi\|_1$ and r:

$$\begin{array}{ll} \min_{\mathbf{c},r,\xi} & r^2 + C\,\|\xi\|_1 \\ \text{subject to} & \|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq r^2 + \xi_i \\ & \xi_i \geq 0,\ \ i = 1, \ldots, m \end{array}$$

Novelty detection

Dual optimisation maximise

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i \kappa\left(\mathbf{x}_i, \mathbf{x}_i\right) - \sum_{i,j=1}^{m} \alpha_i \alpha_j \kappa\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

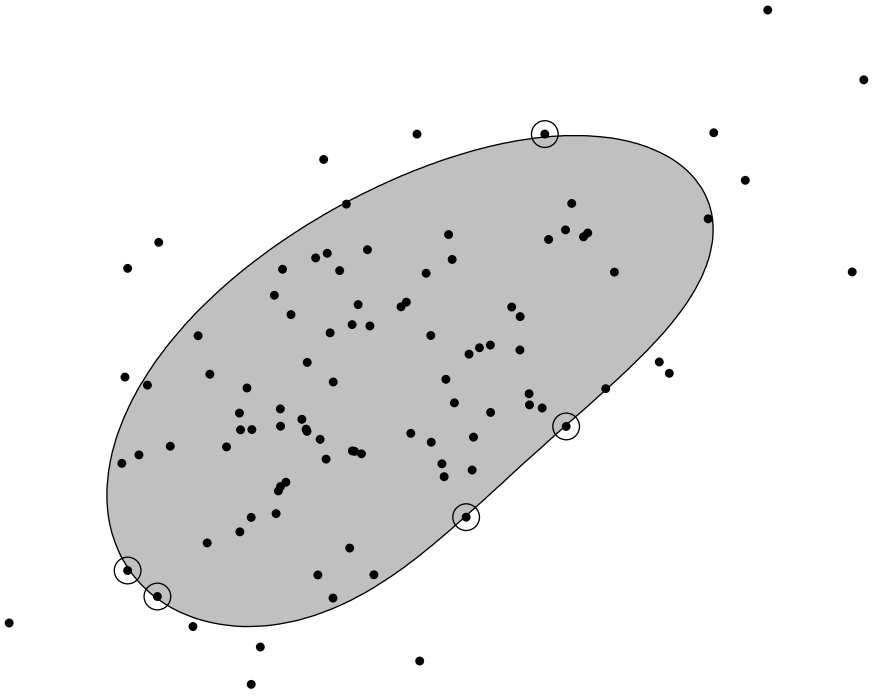
subject to $\sum_{i=1}^{m} \alpha_i = 1$ and $0 \leq \alpha_i \leq C, i = 1, \ldots, m.$
with final novelty test being:

$$f(\cdot) = \mathcal{H}\left[\kappa\left(\cdot, \cdot\right) - 2\sum_{i=1}^{m} \alpha_i^* \kappa\left(\mathbf{x}_i, \cdot\right) + D\right]$$

where

$$D = \sum_{i,j=1}^{m} \alpha_i^* \alpha_j^* \kappa\left(\mathbf{x}_i, \mathbf{x}_j\right) - \left(r^*\right)^2 - \gamma$$

Novelty detection



OVERALL STRUCTURE

Day 1: Introduction to the Kernel methods approach.

Day 2: Projections and subspaces in the feature space.

Day 3: Other learning algorithms with the example of Support Vector Machines.

Day 4: Kernel design strategies.

Part 4 structure

- Kernel design strategies.
- Kernels for text and string kernels.
- Kernels for other structures.
- Kernels from generative models.

Kernel functions

- Already seen some properties of kernels:
  - symmetric:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \langle \phi(\mathbf{z}), \phi(\mathbf{x}) \rangle = \kappa(\mathbf{z}, \mathbf{x})$$

  - kernel matrices psd:

$$\begin{aligned} \mathbf{u}'\mathbf{K}\mathbf{u} &= \sum_{i,j=1}^{m} u_i u_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\ &= \left\langle \sum_{i=1}^{m} u_i \phi(\mathbf{x}_i), \sum_{j=1}^{m} u_j \phi(\mathbf{x}_j) \right\rangle \\ &= \left\| \sum_{i=1}^{m} u_i \phi(\mathbf{x}_i) \right\|^2 \geq 0 \end{aligned}$$

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.

- Note that this is equivalent to all eigenvalues non-negative – recall that eigenvalues of the kernel matrix measured the sum of the squares of the projections onto the eigenvector.

- If we have uncountable domains should also have continuity, though there are exceptions to this as well.

Kernel functions

Proof outline:

- Define feature space as class of functions:

$$\mathcal{F} = \left\{ \sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \cdot) : m \in \mathbb{N}, \mathbf{x}_i \in X, \alpha_i \in \mathbb{R}, i = 1, \ldots, m \right\}$$

- Linear space

- embedding given by

$$\mathbf{x} \longmapsto \kappa(\mathbf{x}, \cdot)$$

Kernel functions

- inner product between

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \text{ and } g(\mathbf{x}) = \sum_{i=1}^{n} \beta_i \kappa(\mathbf{z}_i, \mathbf{x})$$

  defined as

$$\langle f, g \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j \kappa(\mathbf{x}_i, \mathbf{z}_j) = \sum_{i=1}^{m} \alpha_i g(\mathbf{x}_i) = \sum_{j=1}^{n} \beta_j f(\mathbf{z}_j),$$

- well-defined
- $\langle f, f \rangle \geq 0$ by psd property.

Kernel functions

- so-called reproducing property:

$$\langle f, \phi(\mathbf{x}) \rangle = \langle f, \kappa(\mathbf{x}, \cdot) \rangle = f(\mathbf{x})$$

- implies that inner product corresponds to function evaluation – learning a function corresponds to learning a point being the weight vector corresponding to that function:

$$\langle \mathbf{w}_f, \phi(\mathbf{x}) \rangle = f(\mathbf{x})$$

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geq 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\phi(\mathbf{x}),\phi(\mathbf{z}))$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{B}\mathbf{z}$.

Kernel constructions

Following are also valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = p(\kappa_1(\mathbf{x}, \mathbf{z}))$, for p any polynomial with positive coefficients.
- $\kappa(\mathbf{x}, \mathbf{z}) = \exp(\kappa_1(\mathbf{x}, \mathbf{z}))$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \exp(-\left\|\mathbf{x} - \mathbf{z}\right\|^2 / (2\sigma^2))$.

Proof of third: normalise the second kernel:

$$\frac{\exp(\langle \mathbf{x}, \mathbf{z} \rangle / \sigma^2)}{\sqrt{\exp(\left\|\mathbf{x}\right\|^2 / \sigma^2) \exp(\left\|\mathbf{z}\right\|^2 / \sigma^2)}} = \exp\left(\frac{\langle \mathbf{x}, \mathbf{z} \rangle}{\sigma^2} - \frac{\langle \mathbf{x}, \mathbf{x} \rangle}{2\sigma^2} - \frac{\langle \mathbf{z}, \mathbf{z} \rangle}{2\sigma^2}\right)$$

$$= \exp\left(-\frac{\left\|\mathbf{x} - \mathbf{z}\right\|^2}{2\sigma^2}\right).$$

Subcomponents kernel

For the kernel $\langle \mathbf{x}, \mathbf{z} \rangle^s$ the features can be indexed by sequences

$$\mathbf{i} = (i_1, \ldots, i_n), \sum_{j=1}^{n} i_j = s$$

where

$$\phi_{\mathbf{i}}(\mathbf{x}) = x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$$

A similar kernel can be defined in which all subsets of features occur:

$$\phi : \mathbf{x} \mapsto \left(\phi_A(\mathbf{x})\right)_{A \subseteq \{1,\ldots,n\}}$$

where

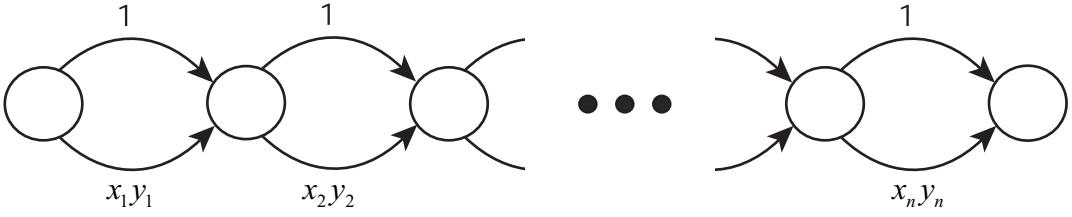
$$\phi_A(\mathbf{x}) = \prod_{i \in A} x_i$$

Subcomponents kernel

So we have

$$\begin{aligned}\kappa_{\subseteq}(\mathbf{x},\mathbf{y}) &= \langle\phi(\mathbf{x}),\phi(\mathbf{y})\rangle \\ &= \sum_{A\subseteq\{1,\ldots,n\}} \phi_A(\mathbf{x})\phi_A(\mathbf{y}) \\ &= \sum_{A\subseteq\{1,\ldots,n\}} \prod_{i\in A} x_i y_i = \prod_{i=1}^{n}(1+x_i y_i)\end{aligned}$$

Can represent computation with a graph:



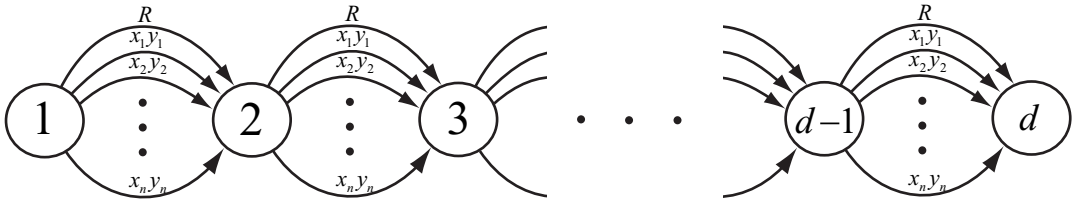
Each path in the graph corresponds to a feature.

Graph kernels

Can also represent polynomial kernel

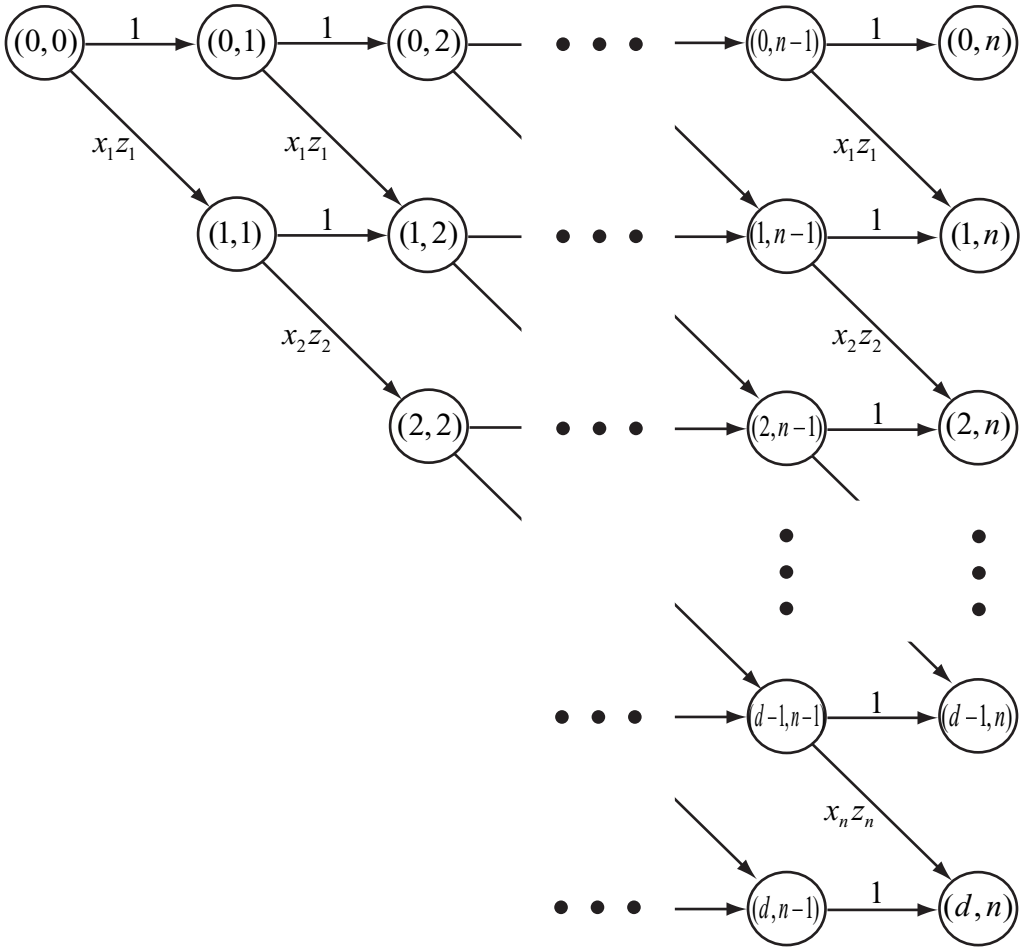
$$\kappa(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + R)^d = (x_1 y_1 + x_2 y_2 + \cdots + x_n y_n + R)^d$$

with a graph:



Graph kernels

The ANOVA kernel is represented by the graph:



ANOVA kernels

Features are all the combinations of exactly d distinct features, while computation is given by recursion:

$$\begin{aligned}
\kappa_0^m(\mathbf{x}, \mathbf{z}) &= 1, \text{ if } m \geq 0, \\
\kappa_s^m(\mathbf{x}, \mathbf{z}) &= 0, \text{ if } m < s, \\
\kappa_s^m(\mathbf{x}, \mathbf{z}) &= (x_m z_m)\kappa_{s-1}^{m-1}(\mathbf{x}, \mathbf{z}) + \kappa_s^{m-1}(\mathbf{x}, \mathbf{z})
\end{aligned}$$

While the resulting kernel is given by

$$\kappa_d^n(\mathbf{x}, \mathbf{z})$$

in the bottom right corner of the graph.

General Graph kernels

- Defined over directed acyclic graphs (DAGs)
- Number vertices $1, \ldots, s$ compatible with edge directions, i.e. $u_i \to u_j \implies i < j$.
- Compute using dynamic programming table DP
- Initialise $\mathrm{DP}\,(1) = 1;$
- for $i = 2, \ldots, s$ compute

$$\mathrm{DP}\,(i) = \sum_{j \to i} \kappa_{(u_j \to u_i)}\,(\mathbf{x}, \mathbf{z})\,\mathrm{DP}\,(j)$$

- result given at output node s: $\kappa(\mathbf{x}, \mathbf{z}) = \mathrm{DP}(s)$.

Kernels for text

- The simplest representation for text is the kernel given by the feature map known as the vector space model

$$\phi : d \mapsto \phi(d) = (\mathrm{tf}(t_1, d), \mathrm{tf}(t_2, d), \ldots, \mathrm{tf}(t_N, d))'$$

  where $t_1, t_2, \ldots, t_N$ are the terms occurring in the corpus and $\mathrm{tf}(t, d)$ measures the frequency of term t in document d.

- Usually use the notation $\mathbf{D}$ for the document term matrix (cf. $\mathbf{X}$ from previous notation).

Kernels for text

- Kernel matrix is given by

$$\mathbf{K} = \mathbf{D}\mathbf{D}'$$

  wrt kernel

$$\kappa(d_1, d_2) = \sum_{j=1}^{N} \mathrm{tf}(t_j, d_1)\mathrm{tf}(t_j, d_2)$$

- despite high-dimensionality kernel function can be computed efficiently by using a linked list representation.

Semantics for text

- The standard representation does not take into account the importance or relationship between words.

- Main methods do this by introducing a 'semantic' mapping $\mathbf{S}$:

$$\hat{\kappa}(d_1, d_2) = \phi(d_1)'\mathbf{S}\mathbf{S}'\phi(d_2)$$

Semantics for text

- Simplest is diagonal matrix giving term weightings (known as inverse document frequency – tfidf):

$$w(t) = \ln \frac{m}{\mathrm{df}(t)}$$

- Hence kernel becomes:

$$\kappa(d_1, d_2) = \sum_{j=1}^{N} w(t_j)^2 \mathrm{tf}(t_j, d_1) \mathrm{tf}(t_j, d_2)$$

Semantics for text

- In general would also like to include semantic links between terms with off-diagonal elements, eg stemming, query expansion, wordnet.

- More generally can use co-occurrence of words in documents:

$$\mathbf{S} = \mathbf{D}'$$

so

$$(\mathbf{S}\mathbf{S}')_{ij} = \sum_d \mathrm{tf}(i,d)\mathrm{tf}(j,d)$$

Semantics for text

- Information retrieval technique known as latent semantic indexing uses SVD decomposition:

$$\mathbf{D}' = \mathbf{U}\mathbf{\Sigma}\mathbf{V}'$$

so that

$$d \mapsto \mathbf{U}_k' \phi(d)$$

which is equivalent to peforming kernel PCA to give latent semantic kernels:

$$\tilde{\kappa}(d_1, d_2) = \phi(d_1)' \mathbf{U}_k \mathbf{U}_k' \phi(d_2)$$

String kernels

- Consider the feature map given by

$$\phi_u^p(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|$$

for $u \in \Sigma^p$ with associated kernel

$$\kappa_p(s,t) = \sum_{u \in \Sigma^p} \phi_u^p(s)\phi_u^p(t)$$

String kernels

- Consider the following two sequences:

$$s = \texttt{"statistics"}$$
$$t = \texttt{"computation"}$$

The two strings contain the following substrings of length 3:

"sta", "tat", "ati", "tis",
"ist", "sti", "tic", "ics"
"com", "omp", "mpu", "put",
"uta", "tat", "ati", "tio", "ion"

and they have in common the substrings "tat" and "ati", so their inner product would be $\kappa_3(s,t) = 2$.

Trie based p-spectrum kernels

- Computation organised into a trie with nodes indexed by substrings – root node by empty string ϵ.
- Create lists of substrings at root node:

  $$L_s(\epsilon) = \{(s(i : i + p - 1), 0) : i = 1, |s| - p + 1\}$$

  Similarly for t.
- Recursively through the tree: if $L_s(v)$ and $L_t(v)$ both not empty:
  for each $(u, i) \in L_*(v)$ add $(u, i + 1)$ to list $L_*(vu_{i+1})$
- At depth p increment global variable kern initialised to 0 by $|L_s(v)||L_t(v)|$.

Gap weighted string kernels

- Can create kernels whose features are all substrings of length p with the feature weighted according to all occurrences of the substring as a subsequence:

ϕ	ca	ct	at	ba	bt	cr	ar	br
cat	λ^2	λ^3	λ^2	0	0	0	0	0
car	λ^2	0	0	0	0	λ^3	λ^2	0
bat	0	0	λ^2	λ^2	λ^3	0	0	0
bar	0	0	0	λ^2	0	0	λ^2	λ^3

- This can be evaluated using a dynamic programming computation over arrays indexed by the two strings.

Tree kernels

- We can consider a feature mapping for trees defined by

$$\phi : T \longmapsto (\phi_S(T))_{S \in I}$$

  where I is a set of all subtrees and $\phi_S(T)$ counts the number of co-rooted subtrees isomorphic to the tree S.

- The computation can again be performed efficiently by working up from the leaves of the tree integrating the results from the children at each internal node.

- Similarly we can compute the inner product in the feature space given by all subtrees of the given tree not necessarily co-rooted.

Probabilistic model kernels

- There are two types of kernels that can be defined based on probabilistic models of the data.

- The most natural is to consider a class of models index by a model class M: we can then define the similarity as

$$\kappa(x, z) = \sum_{m \in M} P(x|m)P(z|m)P_M(m)$$

  also known as the marginalisation kernel.

- For the case of Hidden Markov Models this can be again be computed by a dynamic programming technique.

Probabilistic model kernels

- Pair HMMs generate pairs of symbols and under mild assumptions can also be shown to give rise to kernels that can be efficiently evaluated.

- Similarly hidden tree generating models of data, again using a recursion that works upwards from the leaves.

Fisher kernels

Fisher kernels are an alternative way of defining kernels based on probabilistic models.

- We assume the model is parametrised according to some parameters: consider the simple example of a 1-dim Gaussian distribution parametrised by μ and σ:

$$M = \left\{ P(x|\theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) : \theta = (\mu, \sigma) \in \mathbb{R}^2 \right\}.$$

- The Fisher score vector is the derivative of the log likelihood of an input x wrt the parameters:

$$\log \mathcal{L}_{(\mu,\sigma)}(x) = -\frac{(x-\mu)^2}{2\sigma^2} - \frac{1}{2}\log(2\pi\sigma).$$

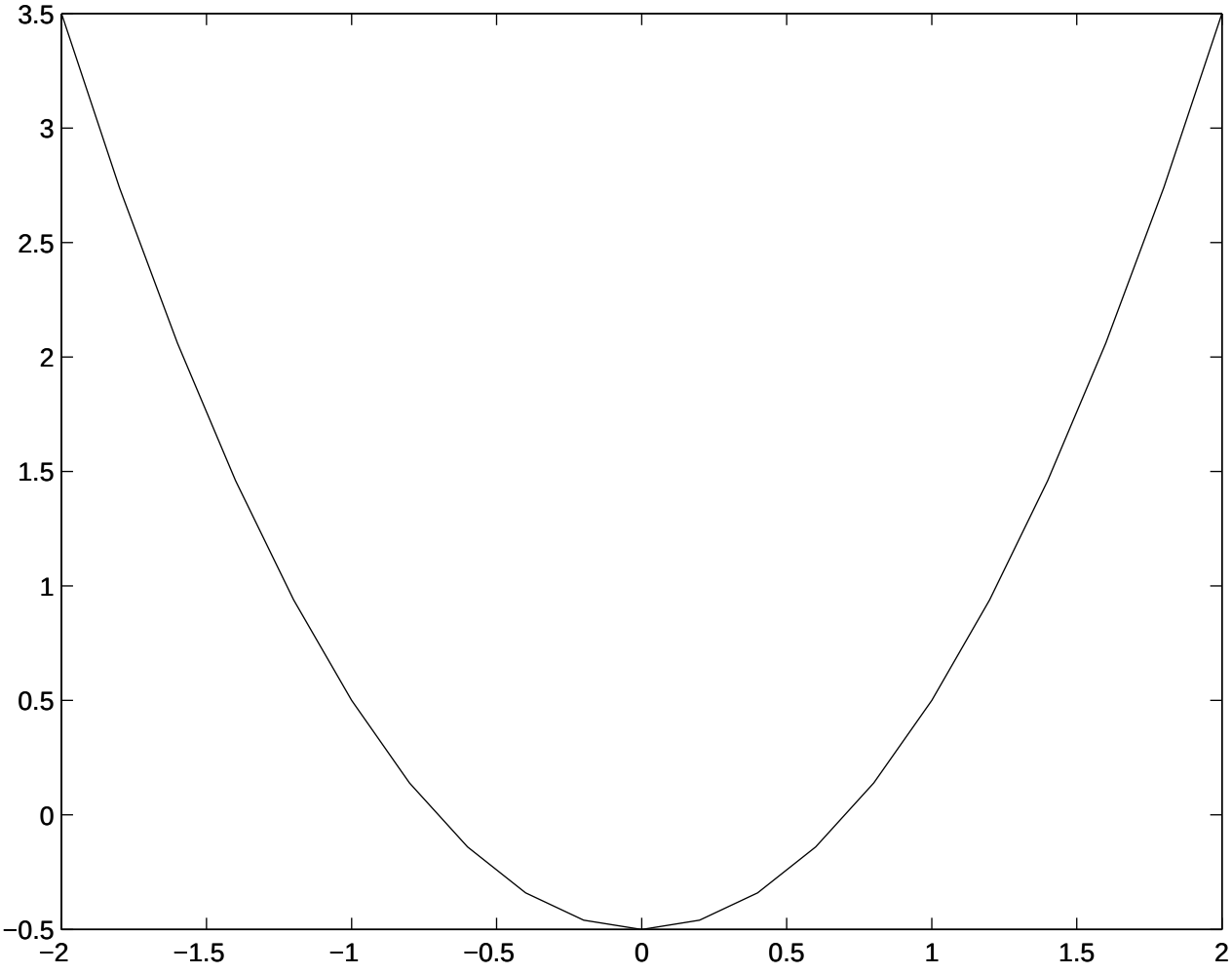
Fisher kernels

- Hence the score vector is given by:

$$\mathbf{g}\left(\theta^0, x\right) = \left(\frac{(x-\mu_0)}{\sigma_0^2}, \frac{(x-\mu_0)^2}{\sigma_0^3} - \frac{1}{2\sigma_0} \right).$$

- Taking $\mu_0 = 0$ and $\sigma_0 = 1$ the feature embedding is given by:

Fisher kernels



Fisher kernels

Can compute Fisher kernels for various models including

- ones closely related to string kernels
- mixtures of Gaussians
- Hidden Markov Models

Conclusions

Kernel methods provide a general purpose toolkit for pattern analysis

- kernels define flexible interface to the data enabling the user to encode prior knowledge into a measure of similarity between two items – with the proviso that it must satisfy the psd property.

- composition and subspace methods provide tools to enhance the representation: normalisation, centering, kernel PCA, kernel Gram-Schmidt, kernel CCA, etc.

- algorithms well-founded in statistical learning theory enable efficient and effective exploitation of the high-dimensional representations to enable good off-training performance.

Where to find out more

Web Sites: `www.support-vector.net` (SV Machines)

`www.kernel-methods.net` (kernel methods)

`www.kernel-machines.net` (kernel Machines)

`www.pascal-network.org`

References

[1] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821 – 837, 1964.

[2] N. Alon, S. Ben-David, N. Cesa-Bianchi, and D. Haussler. Scale-sensitive Dimensions, Uniform Convergence, and Learnability. *Journal of the ACM*, 44(4):615–631, 1997.

[3] M. Anthony and P. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[4] M. Anthony and N. Biggs. *Computational Learning Theory*, volume 30 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.

[5] M. Anthony and J. Shawe-Taylor. A result of Vapnik with applications. *Discrete Applied Mathematics*, 47:207–217, 1993.

[6] K. Azuma. Weighted sums of certain dependent random variables. *Tohoku Math J.*, 19:357–367, 1967.

[7] P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 43–54, Cambridge, MA, 1999. MIT Press.

[8] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network.

IEEE Transactions on Information Theory, 44(2):525–536, 1998.

[9] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[10] S. Boucheron, G. Lugosi, , and P. Massart. A sharp concentration inequality with applications. *Random Structures and Algorithms*, pages vol.16, pp.277–292, 2000.

[11] O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[12] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

[13] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Eurocolt '95*, pages 23–37. Springer-Verlag, 1995.

[14] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Stat. Assoc.*, 58:13–30, 1963.

[15] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[16] V. Koltchinskii and D. Panchenko. Rademacher processes and bounding the risk of function learning. *High Dimensional Probability II*, pages 443 – 459, 2000.

[17] J. Langford and J. Shawe-Taylor. PAC bayes and margins. In *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.

[18] M. Ledoux and M. Talagrand. *Probability in Banach Spaces: isoperimetry and processes*. Springer, 1991.

[19] C. McDiarmid. On the method of bounded differences. In 141 London Mathematical Society Lecture Notes Series, editor, *Surveys in Combinatorics 1989*, pages 148–188. Cambridge University Press, Cambridge, 1989.

[20] R. Schapire, Y. Freund, P. Bartlett, and W. Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*,

1998. (To appear. An earlier version appeared in: D.H. Fisher, Jr. (ed.), Proceedings ICML97, Morgan Kaufmann.).

[21] J. Shawe-Taylor, P. L. Bartlett, R. C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

[22] J. Shawe-Taylor and N. Cristianini. On the generalisation of soft margin algorithms. *IEEE Transactions on Information Theory*, 48(10):2721–2735, 2002.

[23] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.

[24] J. Shawe-Taylor, C. Williams, N. Cristianini, and J. S. Kandola. On the eigenspectrum of the gram matrix and its relationship to the operator eigenspectrum. In *Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT2002)*, volume 2533, pages 23–40, 2002.

[25] M. Talagrand. New concentration inequalities in product

spaces. *Invent. Math.*, 126:505–563, 1996.

[26] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[27] V. Vapnik and A. Chervonenkis. Uniform convergence of frequencies of occurence of events to their probabilities. *Dokl. Akad. Nauk SSSR*, 181:915 – 918, 1968.

[28] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[29] Tong Zhang. Covering number bounds of certain regularized linear function classes. *Journal of Machine Learning Research*, 2:527–550, 2002.